Hardware/Software Support for Securing Virtualization in Embedded Systems

Franck Bucheron, Arnaud Tisserand, Louis Rilling

Hardware/Software Support for Securing Virtualization in Embedded Systems

Franck BUCHERON[1,2], Arnaud TISSERAND[3,2] and Louis RILLING[1,2]

[1] DGA-MI
franck.bucheron | louis.rilling@intradef.gouv.fr
[2] IRISA – [3] CNRS – Univ. Rennes 1 – INRIA
arnaud.tisserand@irisa.fr

Abstract. As far as one is concerned by security in virtual embedded systems, one can say that many ideas or implementations exist today but not really as a global solution and not really in the open-source world. Our goal is to bring a security solution up-to-date and new to build-up a solution from the hardware to an end-user application located in a virtual machine (VM).
The idea of this article is to focus on the lack of secure solutions that can help virtualization and communication which can be implemented on new hybrids (Core + FPGA) development platforms. On one side, these boards are featured with processors that do not have virtualization extensions but are powerfull enough to really support hypervisors and their guests. On the other side some virtualization solutions presently exist for ARM processors but they only refer to TrustZone for their (hardware) security. These hybrid boards can offer us more: we have read some recents and up-to-date specifications made by a consortium to help the implementation of hardware security. In this area, FPGA can help in securing virtualization. But we must notice that, for now, all has been made for Intel/AMD architectures and for a lone operating system. Even so, the whole propositions are too complex to be implemented on embedded systems. So, we will have to use some capabilities in hardware development and make software rearrangements to help us to design a functional solution.

Keywords: TCG, TPM, Embedded security, Cryptographic primitives, Virtualization, Embedded systems, Xen hypervisor, Hardware-software codesign

1 Introduction

Up to some years ago, mainly high-performance processors, such as those in servers and desktops, were powerful enough to support *virtualization*. This was mainly used at software level to provide *isolation*, *security services* and *resources sharing*. Today, even processors for embedded systems can also efficiently support virtualization. But on top of isolation and protection against *software attacks*,

embedded systems have also to face *hardware attacks* such as *side channel attacks* (SCAs). Hence, virtualization for embedded systems requires a combined *hardware/software (HW/SW) approach*. In this paper, we first provide an important review and analyze state-of-art solutions. Second, we describe our own solution. We study an embedded virtualization solution based on HW IPs and a SW stack for efficiently protecting the system against SW attacks. Security against SCAs comes from the use of optimized and SCA-protected hardware IPs for primitives such as symmetric/asymmetric cryptography and true random number generation (TRNG). Using this HW/SW approach, we expect better performances, security and lower power consumption compared to software solutions. Our system is implemented in a Zynq-7000 HW/SW platform composed of a processor (ARM Cortex-A9 dual-core) and a FPGA (Artix-7) in the same device. For the SW stack, we use a derived version of the Xen hypervisor. In order to offer efficient, robust and low-power integrity, confidentiality and authentication primitives, we use a minimal list of existing HW IPs: AES (256 bits) for symmetric encryption/decryption, ECC (between 192 and 256 bits), RSA (2048 bits), TRNG, SHA-2 (256 bits). In addition to these HW IPs, we have to add a few other HW blocks: I/O interface between HW and SW, various internal memories (for Platform Configuration Registers (PCR) and keys), control and execution engine (for the HW part of implemented security policies). At the SW level, we adapted a Xen-like hypervisor to our HW architecture. Our SW code links the various virtual machines (VMs), the management VM (Dom0) and the hardware architecture. The proposed HW architecture and SW stack allows the end user to launch VMs where the executed code is trusted (i.e. verified and the user is granted) and the manipulated data are also trusted (i.e. integrity is ensured and only authorized users have access to these data inside the complete HW/SW platform).
In a first part of this document we will look at the state of the art in our domain of research to get a strong idea of what can be done including the threats of these implementations or proposed implementations. In a second part we will explain the choices made for this work and explain the design of the proposed solution (hardware and software). In a third part of this document, we will conclude and present the work still in progress. Finally, we will make plans about future developments.

2 State of the Art

In a global point of view, we will have to look at three domains: hardware, existing software virtualization solutions in the open-source world and concepts of trusted computing which can link both of them. Then, we will study the threats in those domains that one must tackle in our solution.

2.1 FPGA and ARM Cores

To deal with embedded systems, one has two possibilities: ASICs and FPGAs. ASICs are non-programmable but they can use software intensively. They are

included in "ready-to-use" systems on chip (SoCs) with their board support package (BSP) and software distributions the user can immediatly install. Most popular are linux-based ones. We can find different sorts of CPU inside: Intel's Atom, AMD's G-series, Freescale's PowerPC e300, ARM's Cortex-M series or adaptated IP cores for specific usages. These SoCs fit very well for a general use or students researchs. Some of them are designed for special purposes and include some safety/security like cryptographic accelerators, trusted boot capabilities, efuses possibilities. But we don't get all the detailed specifications for these elements and they have few or no programming capacities. At least they can't upgrade their software ou hardware due to weaknesses (newly discovered vulnerabilities, bugged soft or hard implementation and age). On the other side, FPGAs are black boards and have to be implemented from scratch with all the needed IP plus another special software development (i.e the firmware) for the board support package. For a standalone FPGA board an extra softcore processor should be implemented on it. For a combined development, one can plug a FPGA daughter card into a board containing a processor which result in connections that must be secured between the board and the FPGA. This may cause latencies. To help using FPGAs, few years ago, some hybrid platforms have been produced (Arria or Cyclone based from Altera and Artix or Kintex from Zynq-7000 Xilinx). These two boards have approximately the same architecture for the first designs depending upon the exact reference as shown in figure 1.

Equipment	Atlera's CycloneV	Xilinx's Artix-7
Processor	1× 800 MHz dual-core ARM Cortex-A9	1× 800 MHz dual-core ARM Cortex-A9
Connexion between logic and processing	1× 128-bit write AXI4 bus, 1× 128-bit read AXI4 bus, 1× 32-bit Advanced Peripheral Bus (APB) port	4× 32-bit AXI4 bus and 4× 64-bit AXI4 bus
On chip memory	64 ko	256 KB (192 KB for users)
logic elements	up to 110 K	up to 235 K

Fig. 1. Short comparison of two low-cost hybrid systems.

The Zynq-7000 is the only development board immediatly avaliable at low-cost on the market at the end of this state of the art. All the components are present to consider the realization of projects that had never been made before on such a board. They can provide a secure/authenticated boot with Xilinx credentials which can be replaced by user-provided secure boot up to a certain

extent. They also offer partial reconfiguration possibilities even if it must be manually done with the last Vivado version (2014.3) and one has to purchase a special license for it.
On these boards, one can boot the device under several modes but in this work we only consider secure modes (encrypted and signed). And for this boot mode, one have again 4 possibilities (Flash, QSPI, NAND and NOR). The BootROM is on the processing system (PS) side along with the hash of the RSA public key (for authentication) and 256 kB. of on-chip memory (OCM). On the programmable logic (PL) side one find 2 engines: one AES-256 for decryption purposes and one HMAC SHA-256 for authentication purposes plus efuses, battery backed RAM (BBRAM) and configuration registers. The secure flow of booting, on figure 2, depends on two operation. One the PC side, the bootgen tool generates first the HMAC which is followed by AES encryption then *the hash of the partition* will be signed with RSA private key. On the Zynq side, the bootROM code will authenticate the FSBL's partition and the FSBL will then authenticate the partition, decrypt it and authenticate it.

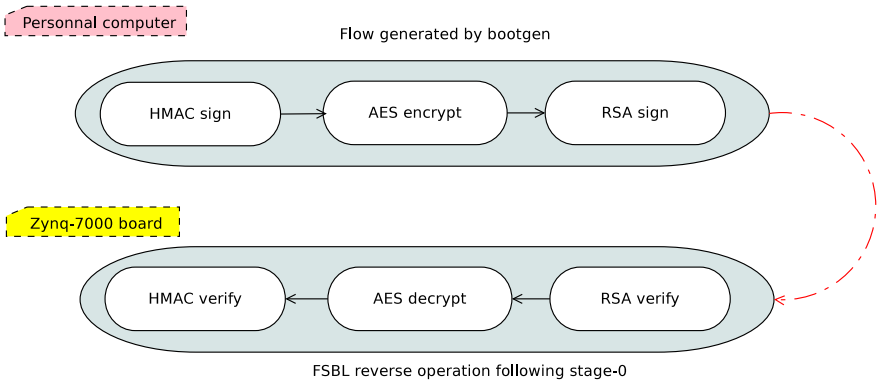

Fig. 2. The complete secure flow for the Xilinx's Zynq-7000 board.[29]

Considering these hybrid boards, both work with an ARM processors designed with 2 Cortex-A9. ARM "secure extension" called TrustZone is present. But, mainly because this extension is "secure", we don't know if the PS of the Zynq-7000 allow playing with such a device and if it does, under which conditions of intellectual property. The Cortex-A9 CPU is the last one of the ARMv7 family which does not have virtualization extensions. New ARM Cortex-A15 and the next-to-come ARMv8 family will have them with a new processor mode helping in hypervisor execution (HYP). This new processor execution mode introduces a 2-walk memory page translation. Another add-on, the large physical address extension (LPAE), will allow to have more than 4 GB of memory but is not avaliable on the Zynq.

2.2 Virtualization on ARM Cores

Virtualization is not a new idea. Its concepts have been well defined in a book: Virtual Machines [21]. Unfortunately, nearly all the functional hypervisors have been made for the INTEL x86 architecture. There are few hypervisors working on ARM cores and very few of them are open-source. One keep closed-source commercial OS out of the scope of our work mainly because their hypervisors are licensed and modifications impossible for a research work.
First of all, one must remind that the used type of virtualization will depend on what ARM core family is used. On ARMv7 family, only software virtualization will be possible. On ARMv7+ (Cortex-A15) and ARMv8 family, both software and hardware virtualization can be considered.
Sierraware [1] has developped Sierra hypervisor which runs on ARM cores. The sources can be downloaded but the Zynq-7000 version has only a binary version. No further implementation (i.e. modifications of kernel sources) will be possible with this hypervisor. Xtratum [2] is an other hypervisor running on ARM platforms but it is very difficult to get the source code. The version we worked on was the 3.2 one and vTPM were implemented on software. Xen ARM (paravirtualized or not) [3] developed a well suited hypervisor for ARM cores but nothing for the Zynq yet. We have no time to implement such a version. An implementation of Xen on an ARM-base smartphone called "secure Xen" has been made in 2008 that shows to be a valuable solution. This document is very helpful in terms of howto [20]. More than this, Samsung designed a new extension on Xen for ARMv7 cores to enable a divided user mode and allow the guest OS be "more isolated" from a guest application [7]. This approach is now superseded by ARM's HYP mode. x-hyp [4] is a new french little hypervisor for ARM world but only runs on QEMU by the time this paper is written. Finally one can see that all these hypervisors do not offer hardware based solutions for a secured virtualization.
We finally found EmbbededXen. A work made by the "Haute Ecole d'Ingénierie et de Gestion du canton de Vaud" (Switzerland)[5] for which the source code is available and runs on a Zynq-7000 platforms [17]. A further implementation has been realized on a HTC desire smartphone. This little hypervisor has great qualities for our works: all the source code (hypervisor, Dom0 and DomU) is included in only one binary file making it easy to upload to the board via NFS or put in flash memory, it is u-boot compatible (so is the Zynq), some security has been deployed on RAM management, hypercalls/syscalls not used in Xen mode and they have initiated a particular mode. The presented work is well mature enough to try to add some new features to it. This project is still in progress and the hypervisor seems easy to manage for prototyping.

[1] http://www.sierraware.com/arm_hypervisor.html
[2] http://www.fentiss.com/en/products/xtratum.html
[3] http://wiki.xenproject.org/wiki/XenARM
[4] http://x-hyp.org
[5] http://www.heig-vd.ch/rad/reds

2.3 Trusted Computing

The trusted computing group (TCG) was primary involved in desktop PC and huge servers but was not really organized for embbeded world and virtualization. Their specifications for virtualization [25] have been published in 2011 and should still been in effect. More than this, virtualization is still considered as an independant domain besides embedded systems. Facing the trend, TCG worked hard to publish a white paper: TPM MOBILE with Trusted Execution Environment for Comprehensive Mobile Device Security [26]. This solution involves greatly the TrustZone and a great part of the trust should depend on it.
But one can't speak about trust computing without mentionning Trusted Platform Modules (TPM) which are the heartbeat of a trusted solution. A version called 1.2 has been widely deployed (over 100's of millions of chips) but is now out-of-age due to scientific improvements on the domain of malicious computing. The up-to-date version is the newly revision 01.07 published in may 2014 which is called "family 2.0" whose total specifications represent more than 1000 pages. The first functional TPM (v1.2) is now obsolete and 2.0 specifications aim at giving more insurance. TPM 1.2 had some limitations and the first one is its age: 2003. In terms of cryptography, only SHA1 and RSA 2048 are fully supported because the US government wanted to avoid *exposed symmetric for export concerns*. To protect confidentiality, RSA2048's key of 112 bits is not strong enough nowadays [11]. To build a policy, this version of TPM had only 3 authorization elements (Authorization value, PCR state and locality) to be combined for a total of 8 possibilities. In terms of target, this version only aimed at desktop PC. The insurance given by the chip could be deactivated from the BIOS by a user. The owner of the TPM controlles all the security and privacy functions.
TPM 2.0 version has an easier way to define security to their "owners". There are 3 separate domains: security (to protect the security of an user), privacy (to expose the identity of the platform/user) and platform (to protect the integrity of the platform services) and for each domain we find proper resources and controls. When one found only one hierarchy for the platform OS which is the user (for the storage) with the former TPM, one has now 2 more: one platform hierarchy to be used by the BIOS and one endorsement hierarchy for the TPM identification. Each hierarchy has its own resources (primary seed to generate EK, enable flag, authorization value, authorization policy). 12 authorization elements and a newborn security policy based on logical AND and OR gives fine grained access control over TPM keys or data like authorization value, locality, PCR state, time limited, duplication, and so on. Some other explanations are presented in this article [12]. Finally, PCRs can use many hash algorithms and doing so, an extension of a PCR must now include the hash and the ID of the used algortihm. Those versions are therefore and, for good, incompatible.
Boot process is the unavoidable base to build a secure chain of trust. First, boot ROM acts as the hardware root of trust for verification (RTV) and forms the start of the root of a transitive chain of verification. The software in the boot ROM locates the first boot phase software in some form of non-volatile storage. The boot ROM could then performs the following sequence of operations:

1. Measures the located image and determines the validity of the measurement,
2. Checks that the version number stored within the image is valid,
3. If all checks have passed, then the device executes the loaded software.

Secure boot is then used only for the early-booted device firmware up to the point of launching the main operating system (OS) boot loader. Whenever an additional module of firmware code (or a data file in some cases) is loaded, the following steps are executed:

1. The code module is measured,
2. The validity of the measurement is determined,
3. If a storage mechanism such as a TPM is available, the measurement is extended into that storage, otherwise it is discarded.

If the Boot ROM is trustworthy, code that fails validation will not be launched during the secure boot process. At last, measured Boot is the process followed *once* a TPM is available during the boot process to carry on the build of the complete trusted process to its end as illustrated in figure 3.
To allow a program to have access to a TPM chip, a software interface must

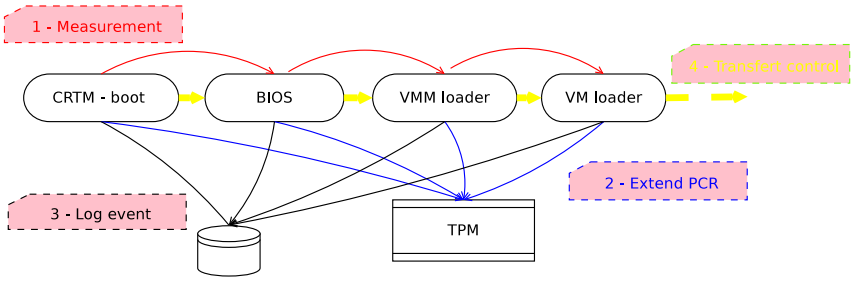

Fig. 3. State-of-Art of a complete secure boot chain.

be provided which is called Trusted Software Stack (TSS) in the TCG terminology. 3 interesting works have been found on this topic. The first one [23] was about a micro-TSS and has been later quoted by the German Federal Office for Information Security (BSI) [1]. For full open-source implementation of a TSS one can follow two projects: Trousers in C language [2] existing since 2005 and jTSS written in java language [3] since 2008. All these works are of utmost interest for their deep study of the mechanisms even if designed for TPM v1.2 family.
It could seem to be a very good ideas to implement all the specifications in motion with the help of a large FPGA but it's impossible in a environment where

[1] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/TSS_Apps/TSS-Apps_en.pdf?__blob=publicationFile
[2] http://trousers.sourceforge.net
[3] http://trustedjava.sourceforge.net

resources are limited. So one has to wonder about their adequation to embedded system. A guide has recently been produced by the US Department of Commerce [10]. This guide can be seen as an "approval" of the TCG specifications because most of its requirements may be satisfied by the TCG. Some other rules can be viewed as an expansion. But in our case, the TPM and its virtual counterparts (vTPM) will have to make a lot adjustments to cope with restrictions on silicon area, energy, power to fullfil the platform constraints. In theory, they seems to be clones of each other and their functionalities are enumerated in figure 4.
Here, we must notice that some european projects try to implement a physical realization of Trust in embedded computing [15] but, in 2010, it was in version 1.2 and recently, in june 2013, by the Institute for Applied Information Processing and Communication (IAIK) of Graz University of Technology, Austria, where a whole TPM v2.0 platform has prototyped on a Spartan3A FPGA [1].

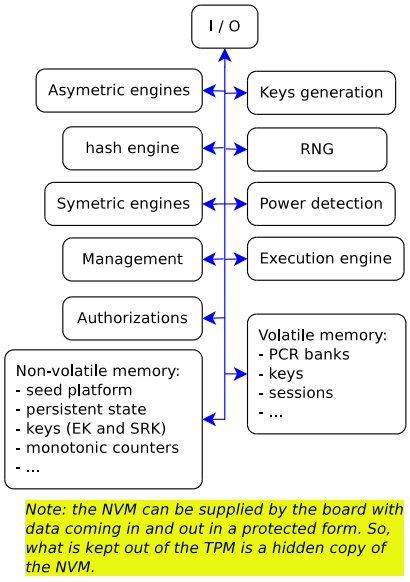

Fig. 4. In concrete terms, TPM and an vTPM should look like that meaning that all the functionalities must be present but may be shared.

For the virtual TPM, a basis solution has been explain in [22]and illustrated in figure 5. This first solution was solely software. Later, other were published. One get the feeling of it and deployed an exclusively hardware-based solution with a vTPM for each VM [24]. One presented a paravirtualized TPM solution which shared TPM and vTPM in only one instance [6]. Another one implement

[1] http://trust2013.sba-research.org/wp-content/uploads/2013/06/Martin-Pirker.pdf

a property based TPM-vTPM solution [19]. But none of these solutions applied to ARM processors and they were too old to get the Zynq-7000 capabilities into account and the new specifications of the TPM v2.0.

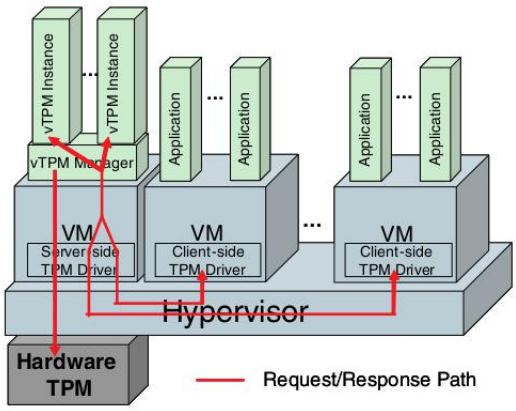

Fig. 5. The mostly implemented solution in para-virtualized hypervisors [22].

2.4 Threats

As one consider threats at the scope of an entire SoC, one will have to look at the whole scope of possible vulnerabilities and later only have a glance at those which could create a vulnerability of our study concerns.
Attackers who can exploit the vunlerabilities of a SoC can be divided into 3 classes: [1] kiddies with little knowledge, engineers with solid knowledge and funds and finally states or organized crime which include terrorism. Their objectives will also be different depending on the range of the power they own (personnal challenge, technologic race or spy/cyberwarfare). For a SoC like Zynq-7000, one must care about 3 categories of vulnerabilities that exists: hardware, FPGA and software (virtualization). Network threats will be less studied because the fact that the network stack is badly coded or the network protocole not respected is not of our concerns.
At the lower level, there are a lot of threats for microelectronic hardware which could also be classified in 3 categories:

1. Backdoor in hardware
2. Hardware vulnerability
3. Hardware assisted malware.

[1] http://watchguardsecuritycenter.com/2013/05/30/hacker-profiles/

Most of these malwares can be found at production level. A malicious modification of hardware is more than possible. It can provide a very good advantage to opponents and worst in the case of security or critical domains. Here the vulnerability is at the Hardware Design Language (HDL). If the firm is fabless risks are greater. The supply chain for integrated circuits (IC) should be trusted from the designers to the end-users. Exploiting hardware vulnerabilities can lean to put a rootkit somewhere on it. For example, the Raksasha malware [1] built-on with free softwares supports about 230 x86 architecture motherboards. This bootkit is possible due to 30 years old mandatory compatibility of the Intel hardware architectures [2]. Updating/upgrading an obsolete hardware is no more an accurate solution.
For FPGAs, there are also sources of trouble. If not protected, a bistream could be counterfeited, cloned or stolen. The resulting effects of these threats, appart from economic and social damage, are that reliability will certainly decrease and security/safety won't be assured anymore. In critical systems like military, aerospace or medical, results can be human death.
About threat on virtualized environments, one must consider that there are two sorts of possible attacks: those that are specific to general purpose computing and those that are specific to virtualization. In this article, we only focus on virtualization-specific threats.
Compromizing the underlying hypervisor can be possible due to a physical vulnerability: on boot, legacy BIOS could be infected by a bad code that could corrupt the following startup sequence. Later, on the hypervisor side, the results can be a compromized hypervisor that has implemented hidden functions, installation of another malicious hypervisor (*example*: blue pill Virtual Machine Monitor (VMM) rootkit) and at last a deactivation of the targeted hypervisor. It would be then possible for an attacker to try to modify one of the VMs before they start.
Managment interfaces can then be the target of an attacker. In case it is running under a well-known vulnerable operating systems (OS), the task should be easier. Threat can also come from network or a compromized guest that could "escape" off its cage.
At upper level, a guest can be the target of attacks from Internet or from an already compromized guest. A simple goal could be to try spying the activity in an hypervisor or another VM. An infected application could do a sequence of instructions that it was not supposed to do inducing a possible leak of informations. These malicious actions conduct malware programmers to adapt their attack technics to this new playing ground: the possibility for a malware to infect all the virtualized platform forces the anti-virus suites to move their product downto the hypervisor. And again, an iterating challenge will be to identify the underlying hypervisor to exploit its vulnerabilities [14] [13].

[1] http://2012.hackitoergosum.org/blog/wp-content/uploads/2012/04/HES-2012-jbrossard_fdemetrescu-Hardware-Backdooring-is-pratical.pdf
[2] http://resources.infosecinstitute.com/hardware-attacks-backdoors-and-electronic-component-qualification/

Finally, a new task specific to virtualization is migration of VMs. This process, from the deactivation, the transfer and the resuming of a VM with all its configuration is a new problem to care about. The attacker could try to replace the guenuine VM by a malicious one. If the blob is not protected enough, he could also try to get some secrets.
A threat to homogeneity of embedded systems has been pointed out due to virtualization as too strong isolation can be a flaw for cooperation between process [1]. This means that forbidding the share of informations between guests/applications can force software designers to employ less secure solutions to get the needed information.
An other approach in ordering malwares have been made in [9]. What is relevant there is that one can again see that researchs had been made on an Intel platform. Nevertheless, ARMv7 cores could possibly suffer from such malwares due to the only CPU mode left for both guests and applications: the user mode. There is a special device in ARM architectures which is called TrustZone and act partly on the hardware side and partly on the software side. The main problem with this secure device is the way it can be programmed. It is not well documented and not easy to play with. Recently vulnerability was detected and presented at the Black Hat USA 2014 conference in [16] and [18] in the HackInParis conference 2013. More that this, when one codes an application in the TrustZone, one must send the package to the motherboard provider to be signed and install at boot. So the keys of our own security are lawfully given to them.
There is also some vulnerabilities in TPM implementations. In some TCG-enabled platforms, communication channels (buses) between TPM, RAM and microprocessor could be unprotected. Moreover, microprocessors could use an external boot ROM to store the Core Root of Trust for Measurement (CRTM). This CRTM is a a piece of executable code starting the measurement of bootstrapping components. It must be an immutable portion of the platform initialization code that executes itself on resets. If the ROM could be switched to inject malicious boot code and compromise the chain of trust, TPM will be of no use at all [5]. Recently, Rakshasa [1] malware proves to inhibit the use of TPM of an Intel platform [3].
Even if trusted computing means confidence through the early powering of the platform to the running of user high level applications, backdoors can be now inserted at gate level and their identification be more and more difficult. And if virtualization is introduced in embedded systems, with many VM, one have to cope with an extended way to carry on guarantee of integrity to all underlying domains. That's why ,three topics have been identified to develop a solution which is aimed to bring more insurance to the virtualization for embedded systems: trusted computing that gives insurance, up-to-date hardware and virtualization which mainly gives isolation.

[1] http://fr.slideshare.net/endrazine/defcon-hardware-backdooring-is-practical

3 Contribution

What we propose in this article is to implement a solution as light as possible, to offer integrity, confidentiality and authorization to an embedded hypervisor, considering TPM and virtual TPM implemented on an Zynq-7000 platform which featuring both a general purpose processor (GPP) an FPGA on it. The objective should consist of (i) an hardware IP which will contain new developments (blocks), reuse of some of the opensource best ideas and as secure as possible against hardware attacks, (ii) hypercalls extensions for the hypervisor and (iii) a software library containing some of the main functions implemented in C and assembly language. This secure solution is intended to act at two levels. The hardware level will have to give trust of the integrity of IP to be plugged onto the programmable logic: this will be an measured boot and not only secure boot. The software level will have to complete and extend the chain of trust to the end-user application: the VMM which must trust the launch of a virtual machine and an application inside a VM has to give guarantee of its integrity to the uperlying operating system or remote system. The expected solution could look like figure 6.

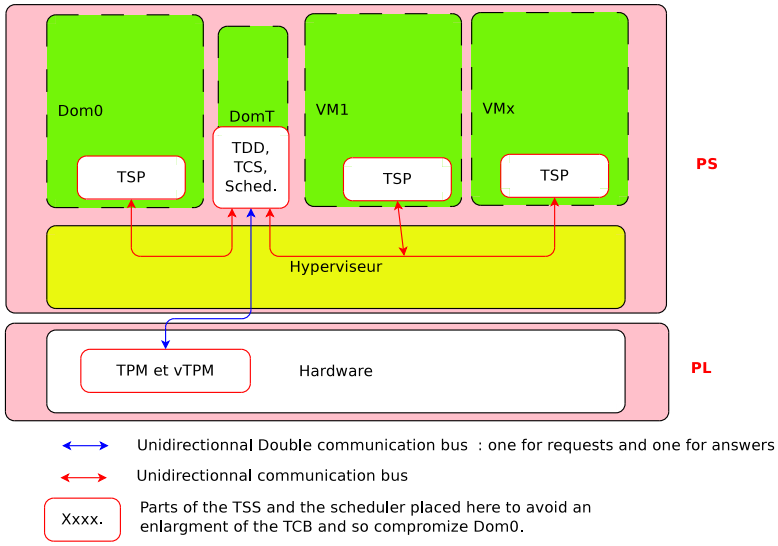

Fig. 6. Overview of the proposed solution. A leightweight hypervisor of Xen type with some VM looking for trust in an FPGA implemented TPM

3.1 Threats Models

As we state previoulsy, we work on an entire platform. The trusted computing base we are trying to define has to protect the system and must be able to protect its own security. See the definition of the trusted computing base in the Orange Book Security [4]. And we can mention that if trust is good, control is better to help reducing threats.
The hardware threats will be mentionned and taken in account only to minmize what actually exists in the state of the art. In fact our studies has to find a solution only for software attacks. We should care only about voltage spikes and frequency shifts (i.e: power faults) that are important for the IP. We will have no action on the ARM Cortex-A9 in terms of making a counter-measure for a possible hole in the ARM Coretx-a9 processor. Likewise, we have no possible action on a BIOS attack resulting in the corruption of it. And finally we won't fight against a possible hardware trojan or backdoor in the Zynq-7000 development board. On an other CPU (OpenSparc) some solutions has been studied in [28] and implemented.
On the software side, we can't do anything against a badly coded application. The hardware solution we propose to help virtualization is to give a proof of integrity that even if the code is bad, this is this bad code that will be executed as asked. The proposed solution will be implemented to give insurance that what is running or about to run will be authorized to be launched. Another insurance will be to try to give a guarantee that the running hypervisor has not been modified since the initialization of the platofrm.
At the HDL level, solutions have been implemented in "Silencing Hardware Backdoors" [28].

3.2 The Hardware Platform

The proposed IP should be built-up with several differents pieces (blocks). Some of the pieces will be a part of our contribution and some picked up from open-source world and/or academic researchs with the permission of their creators. Of course, all the beginning work will consist of a definition of each of the elements, their role, their data in and out. The main work will be the development of the TPM/vTPM IP which will be feeded by a secure boot and which will offer services to upper levels through a software stack.

The sensitive problem of the chain of trust The anchor of trust lean on an secure and measured boot.
In secure boot, each part of loaded code will act as a proxy for the next element to be loaded but it stops until all the checks made success. If a check fails then the boot process stops and the platform is also stopped. The right term, in TCG terminology, is verified boot but secure boot is more widely used: this is made with comparison of signed parts of the boot process. But we have to monitor all the boot process too but in a different way: measurement. First, the component

that measures the initial boot must be trusted. This CRTM is hardware-based. Then, if the loaded piece of software has been measured and proves to be reliable, the component can measure the next software to be loaded, and again there is a solid basis for trust. And this process is iterated for each step, ending with the launching of an user application inside a VM as explain in figure 7.
On the Zynq-7000, however, the very first part of this process (called stage-0)

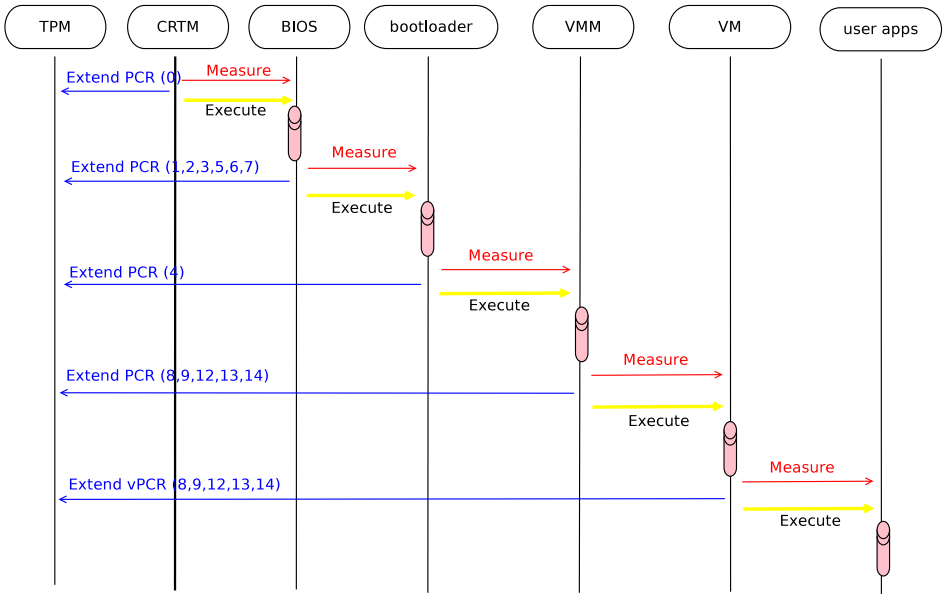

Fig. 7. measured boot: measuring, launching, reporting

is not accessible to the designer and/or the user! Unfortunately, the Zynq-7000 board does not offer the possiblity to modify its BIOS to make a Core Root of Trust. On this point we will have to trust Xilinx because we will have no possible action on the stage-0 boot and we have to make this hypothesis: hope that the first part of the boot chain of the Zynq-7000 (stage-0) is *strong enough* to give us insurance that we can plug our IP securely on the PL. These Xilinx restrictions cannot be bypassed. But later, playing with authentification/encryption, we can implement a large amount of partitions in our first stage boot loader (FSBL) or second stage boot loader (SSBL) to allow the leaning of a rather secure solution. And this solution will be the TPM/vTPM IP for a secured virtualization. This can be seen in Xilinx documentation [29] and [30].
All the measurements made consecutively will be stored, for some parts in an non-volatile memory (NVM) and other parts in volatile memory registers called PCR. As a result, our solution will start on FSBL.

The I/O interface is the first element of the IP that will receive requests from the software part of the systems and send back a result as soon as possible. Two separated buses for in and out flows are a good solution to later develop parallelism. The width of such a bus will be 32 bits to keep fully compatible with the TCG current requirements.
A command received by this interface would have been verified at hypervisors level and carry the id of the VM. The I/O interface will put this command in a scheduler considering its priority (given by the hypervisor) and policy (given by the TPM). One command will be executed at one time and then, this command will "own" all the TPM. This is shown in figure 8.
Before launching the execution of the command, the I/O interface will have to verify that the previous command has ended its work and that the required algorithms are available. If not the case, partial reconfiguration will load the correct bitstream after having verified its integrity at last.
Some different operations will successively follow the following algorithm 1

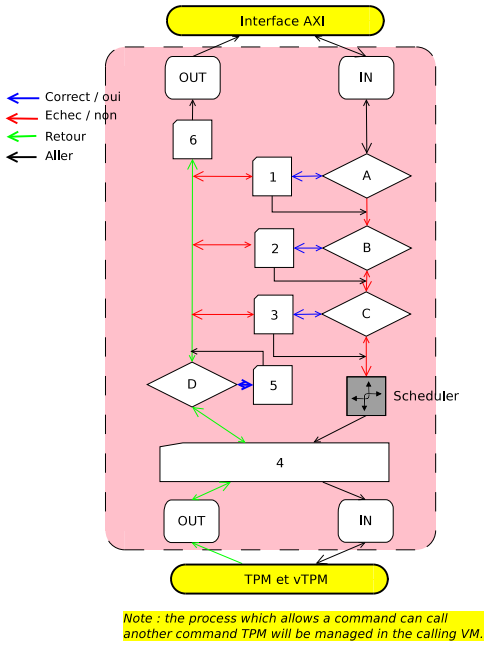

Fig. 8. An important key to TPM functionnality

The internal bus of the TPM/vTPM will be shorter than the original one due to the center position of the execution engine. The width of these buses only need to match the width of a TPM2 command, so 32 bits. In fact, this layout

Algorithm 1 Managment of a TPM2 command issued from the VMM ou VM but transmitted by DomT

Require: On reception of a valid TPM2 command
  if (command header is correct) **AND** (handles exist) **AND** (authorizations are valid) **then**
    make it execute
    wait for aknowledge
    create the return header
  else
    send error with error_id
    report error
  end if

has been chosen considering that having shorter buses is better a long one in terms of side channel analysis aand and to avoid latencies.

The execution engine. One wonder in the case of programmable logic if it would not be more interesting to think about an alternative layout for our TPM-vTPM in a form of a star network. This will move the bulk of work to this unit knowing that only one command coming from VMM or a VM owns the TPM at one time (only one state is possible). The figure 9 also shows the I/O interface will now only interact with one element and not all the possible element in a TPM.

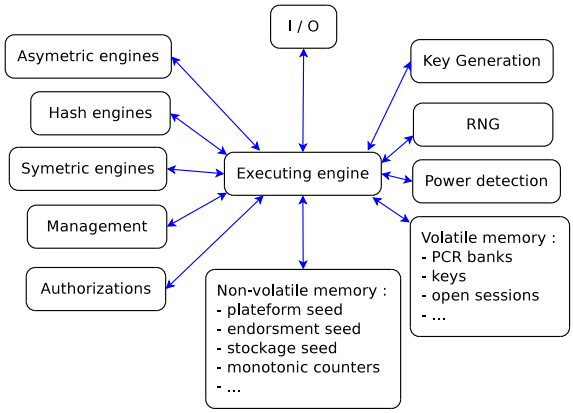

Fig. 9. In such a configuration, we clearly see that the execution engine is at the crossroads between TPM and vTPM

The symmetric and asymmetric crypto engines. There are 2 sorts of engines and TCG specifications explain that more than one engine can be deployed in TPM. As place on the Zynq-7000 is not extensible, we have to consider partial reconfiguration. Bitstreams of each algorithms will have to be shipped with the FSBL and installed in non-volatile memory to be called as necessary. At this point, we must notice that Zynq-7000 has already a RSA2048 algorithm for authentication and an AES256/HMAC algorithm but only for decryption.
The algorithms we plan to use already exist and will be adapted to our developments. Ellyptic curve cryptography (ECC), of course is mandatory. Some of the other cores could be found on opencores website [1] for example.
In our bitstreams, care will have be taken to reduce vulnerabilities of our designs to avoid information leakage [8]. For this, flexibilty of SRAM FPGAs has proved to be of a great advantages [2] to quickly upgrade, many times, a modified version of an IP.

The PCR. The first 8 PCRs (number 0 to 7) will be used by all virtual PCRs (vPCR) because they represent the platform secure state, so only one bank of these PCR will be created in an read-only mode for the VM. The 16 others will represent the VM personnal measures. See figure 10 for explanations. Hash algorithm ID will have to be added to the measurement on each PCR to reflect the property that a PCR can choose from more than one hash algorithm. TPM PCR values will also have to be available both locally and remotely for attestation. For example, a TPM will be able to digitally sign the PCR value to help remote attestation and avoid tampering, corruption, and malicious software.
The key management for a TPM, depicted in figure 11, is not a public key infrastrucure (PKI) as one can have in large systems. In fact, it will have to be as reduced as possible considering that we are in an embedded system. The Endorsement key (EK) will be kept and the Storage Root Key (SRK) protected in NVM because they are the basis of all future credential extensions. The Attestation Identity Key (AIK) derived from EK will be stored on a volatile memory, inside our TPM (but encrypted). AIK will be used to sign data and an EK Credential. It is in fact a X.509 v3 certificate designed to issue credentials for other restricted signing keys (Attestation Keys) [27]. So, this credential will be used to quote (validate) AIK of VM. All the keys (except for the TPM) will be created in our TPM and will be independant from each other for migration purpose.

the Random Number Generation (RNG) has two use cases in a TPM: internally, generate keys and externally generate key using software or feed an pseudo-RNG. That's why, one of the uses of a TPM is to feed /dev/random on a linux box. These utilizations will be preserved and extended to the use by vTPM.

[1] www.opencores.org

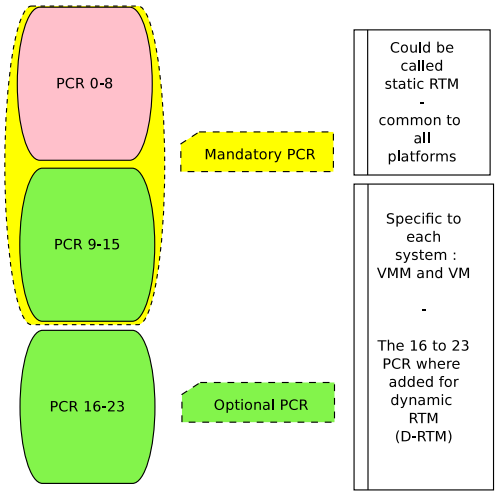

Fig. 10. a VMM will include one READ-ONLY link to the original 0 to 8 PCR bank and a dedicated 9-23 PCR bank

3.3 The hypervisor

We will work on a fork of EmbeddedXen. We know that specifications from the TCG are to be followed in differents domains of this consortium if we look for a minimum of compatibility. But, too much specifications deployed on a small part of silicium would result in non-sense developments. Our work will be to get the specifications we think to be of a prime importance for security knowing that all of them will not be implemented on a SoC (for example: whole policies specifications on a small embbeded system whose configurations are often static).
We will try to respect as far as possible recommandations from XenProject to implement security on the Zynq-7000 board. For example: all extra-services (even security) should not not be implemented in Dom0 but must have a little VM dedicated to them. This is done to not enlarge the Trusted Computing Base (TCB) which is the area of code exposed to attackers.
We also have to develop some hypercalls to catch all the requests to the TPM/vTPM send by the VM but also by the VMM in the case of launching a VM and realize an integrity check before.
This little VM will only manage the demand/response to/from the TPM. It will provide some little services like checking the validity of a command, avoiding deny of service (DOS) attack to the TPM by counting the number of sendings from a VM, software attestation of the source (signing with a dedicated private key in the VM and attesting with its public counterpart in the I/O interface of the TPM), adding an predefined ID of the callling VM to the send command for further identification and management of the request/answers, for a short list.
All the TPM2 commands will not be implemented (due to lack of time and may

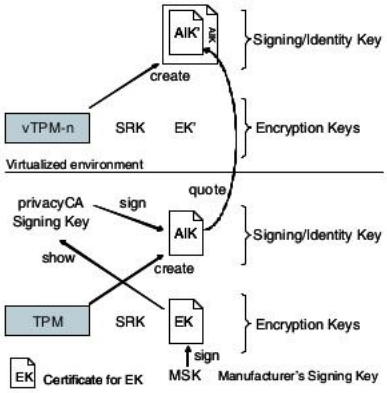

Fig. 11. Key managment as explained in [22]

be place on the silicon area). Basic operations will first be done followed later by more evoluted ones. Here, we must notice that all TPM2 commands have a description in C language which helps for rapid implementation.
The TSS offered to the VM will not be as detailled as it should be for a complete implementation. Two distinct parts will be implemented in our DomT: One in kernel mode called trusted device driver (TDD) et one in user-mode called trusted core services (TCS). TDD which run in kernel mode will be implemented as the only link between the TPM and the rest of the platform (including the VM managment Dom0). In the VM, the trusted service provider (TSP) will be the only access point for a TPM/vTPM request. One can again see that the DomT is at the crossroads between software and hardware in the figure 6 of this article.

4 Conclusion and future works

There is two parts in our works. One leans on hardware and have to code into VHDL plus firmware in C/assembly language. One leans on software and offer services upon an already existing but modified hypervisor.
We think this solution is an interesting one by the way the hypervisor can be modified wihout side effects to the trust offered at different levels. The IP can also be modified and the hypervisor will not be impacted by theses evolutions. The supporting Zynq-7000 board has extended capacities (ASIC/FPGA) that allows this solution should ideally be a "plugin". One great challenge is to make a complete system where latencies will not be a major drawback. One other point to care about is the size of the IP. There may be some trade-offs in implementations to make there.
The TPM-vTPM IP is the most heavy work to realize, the basis of all future

work that could be further imagined. The first bloc to be initiated is the I/O interface, then will follow the internal bus, the execution engine, the PCR/vPCR, the crypto-engines and so on.
At upper software level, little changes are to be made to the chosen hypervisor. At first a software stack to allow developpers to use hardware IP services for each VM. And second, the implementation of a small VM dedicated to the managment of all the transactions with the TPM/vTPM. The last part of the work belongs to the software engineer in the use the avaliable calls to the TPM.

References

1. Aguiar, A., Hessel, F.: Embedded system's virtualization : The next challenge ? In: Rapid System prototyping (RSP), 2010 21st IEEE International symposium on. pp. 1–7. IEEE (2010)
2. Bossuet, L., Gogniat, G., Burleson, W.: Dynamically configurable security for sram fpga bitstreams pp. 73–85 (2006)
3. Brossard, J.: Hardware ackdooring is practical (2012), `http://www.toucan-system.com/research/blackhat2012\_brossard\_hardware\_backdooring.pdf`
4. Department of Defense, USA: Orange security book (1983), `http://csrc.nist.gov/publications/history/dod85.pdf`
5. Eisenbarth, T., Güneysu, T., Paar, C., Sadeghi, A.R., Schellekens, D., arko, W.: Reconfigurable trusted computing in hardware. In: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing. pp. 15–20. STC '07, ACM, New York, NY, USA (2007)
6. England, P., Loeser, J.: Para-virtualized tpm sharing pp. 119–132 (2008)
7. Joo-Young, H., Sang-Bum, S., Sung-Kwan, H., Chan-Ju, P., Jae-Min, R., Seong-Yeo, P., Chul-Ryun, K.: Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In: CCNC IEEE Consumer Communications and Networking Conference. pp. 257–261 (2008)
8. Kastner, R., Huffmire, T.: Threats and challenges in reconfigurable hardware security. In: Proceedings of the 2008 International Conference on Engineering of Reconfigurable Systems & Algorithms, ERSA 2008, Las Vegas, Nevada, USA, July 14-17, 2008. pp. 334–345 (2008)
9. Murakami, J.: A hypervisor ips based on hardware assisted virtualization technology (2008), `http://www.blackhat.com/presentations/bh-usa-08/Murakami/bh\_us\_08\_murakami\_Hypervisor\_IPS.pdf`
10. Nationale Institute of Standards and Technology: Guidelines on hardware-rooted security in mobile devices (draft) (2012), `http://csrc.nist.gov/publications/PubsDrafts.html#SP800-164`
11. Nationale Institute of Standards and Technology: Recommandation for key management : Part 1 : General (revision 3) (2012), `http://csrc.nist.gov/nistpubs/800-57/sp800-57_part1_rev3.pdf`
12. Osborn, J.D., Challenger, D.C.: Trusted platform module evolution. APL technical Digest 32 (2013)
13. Pearce, M., Zeadally, S., Hunt, R.: Virtualization: Issues, security threats, and solutions. ACM Comput. Surv. 45(2), 17:1–17:39 (2013)

14. Pék, G., Buttyán, L., Bencsáth, B.: A survey of security issues in hardware virtualization. ACM Comput. Surv. 45(3), 40:1–40:34 (2013)
15. Ponsini, N.: Trusted embedded computing http://cordis.europa.eu/project/rcn/85362_en.html
16. Rosenberg, D.: Reflections on trusting trustzone (2014), https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf
17. Rossier, D.: Embeddedxen : A revisited architecture of the xen hypervisor to support arm-based embedded virtualization (2012), http://sourceforge.net/projects/embeddedxen/
18. Roth, T.: Next generation mobile rootkits (2013), https://www.hackinparis.com/sites/hackinparis.com/Slidesthomasroth.pdf
19. Sadeghi, A.R., Stble, C., , Winandy, M.: Property based tpm virtualization. In: 11th International Conference, ISC 2008, Taipei, Taiwan, September 15-18, 2008. pp. 1–16. Springer-Verlag, Berlin, Heidelberg (2008)
20. Samsung Electronics Co, ltd: Secure xen on arm user's guide (2008), http://downloads.xenproject.org/Wiki/XenARM/Secure_Xen_on_ARM_User_Guide_v1_0.pdf
21. Smith, J.E., Nair, R.: Virtual machines. Elsevier (2005)
22. Stefan, B., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vtpm: Virtualizing the trusted platform module. In: Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15. USENIX-SS'06, USENIX Association, Berkeley, CA, USA (2006)
23. Stuble, C., Zaerin, A.: tss a simplified trusted software stack 6101, 124–140 (2010)
24. Stumpf, F., Eckert, C.: Enhancing trusted platform modules with hardware-based virtualization techniques. In: Proceedings of the Second International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2008, August 25-31, 2008, Cap Esterel, France. pp. 1–9 (2008)
25. Trusted Computing Group: Virtualized trusted platform architecture specification (2011), http://www.trustedcomputinggroup.org/resources/virtualized_trusted_platform_architecture_specification
26. Trusted Computing Group: Tpm mobile with trusted execution environment for comprehensive mobile device security (2012), http://www.trustedcomputinggroup.org/files/static_page_files/5999C3C1-1A4B-B294-D0BC20183757815E/TPM%20MOBILE%20with%20Trusted%20Execution%20Environment%20for%20Comprehensive%20Mobile%20Device%20Security.pdf
27. Trusted Computing Group: Credential_profile_ek_v2.0_r12_publicreview (2014), http://www.trustedcomputinggroup.org/files/static_page_files/DCD56924-1A4B-B294-D0CEF64E80CEE01E/Credential_Profile_EK_V2.0_R12_PublicReview.pdf
28. Waksman, A., Sethumadhavan, S.: Silencing hardware backdoors. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy. pp. 49–63. SP '11, IEEE Computer Society, Washington, DC, USA (2011)
29. Xilinx: Secure boot of zynq-7000 all programmable soc (2013), http://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf
30. Xilinx: Zynq-7000 all programmable soc secure boot (2014), http://www.xilinx.com/support/documentation/user_guides/ug1025-zynq-secure-boot-gsg.pdf